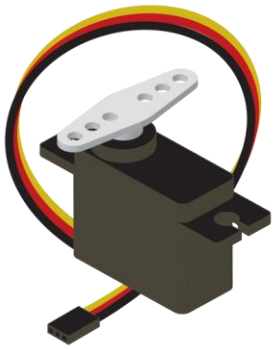
Interface Servo motor with STM32

I have already covered a tutorial about Pulse Width Modulation in STM32 Tutorial 14 and in this tutorial, I am going to cover one of its applications. Today we will see how to control servo motor with STM32 by using PWM.



Servo motors use feedback to determine the position of the shaft, you can control that position very precisely. As a result, servo motors are used to control the position of objects, rotate objects, move the legs, arms, or hands of robots, move sensors, etc. with high precision.

Most servo motors have the following three connections:

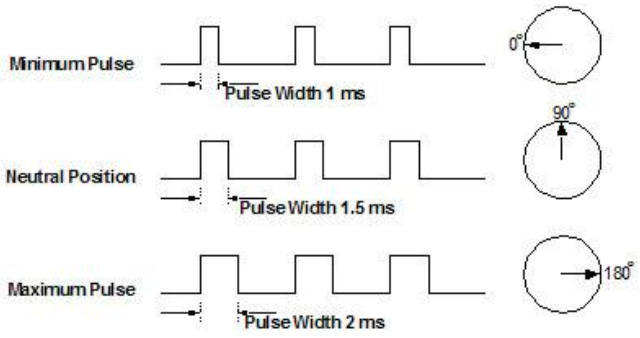
- Black/Brown ground wire.
- Red power wire (around 5V).
- Yellow or White PWM wire.

HOW IT WORKS

Servos can go from 0 to 180 degrees depending on the width of the pulse. You need to keep the pulse (+5v) high for a particular amount of time. A few are listed below:

- 1 milliseconds and corresponds to 0 degrees
- 1.5 milliseconds and corresponds to 90 degrees
- 2 milliseconds and corresponds to 180 degrees

The period between two pulses must be 20ms or we can say the frequency of the pulse must be 50 Hz

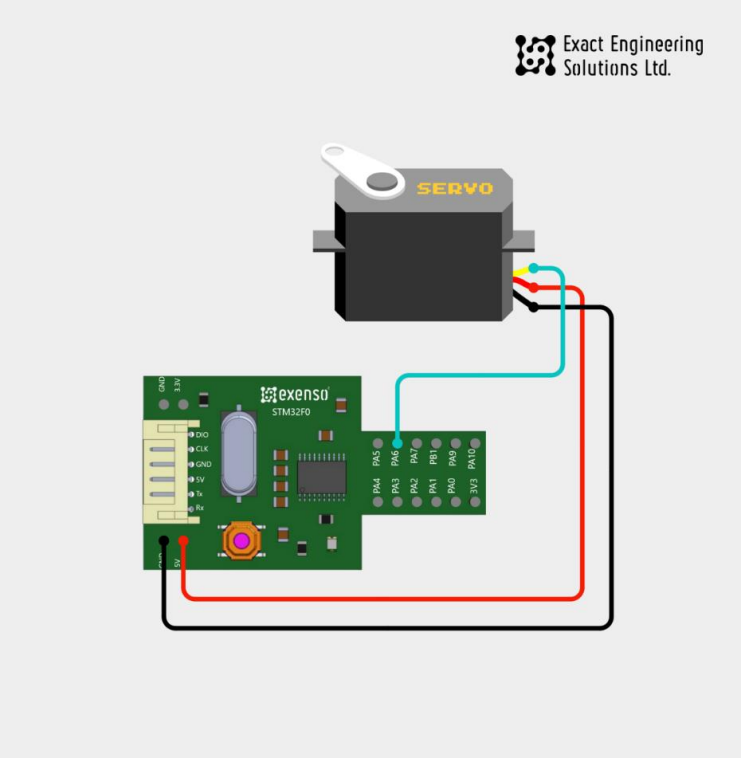


Components Required

You will need the following components −

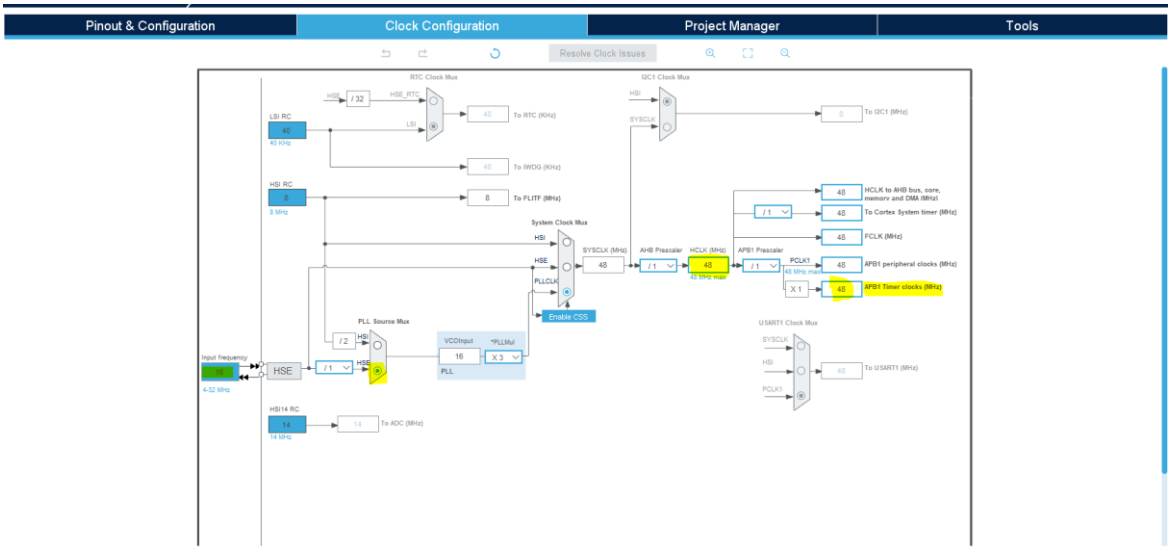
- 1 × Breadboard
- 1 × STM32F030F4P6
- 1 × Servo Motor
- Some Jumper

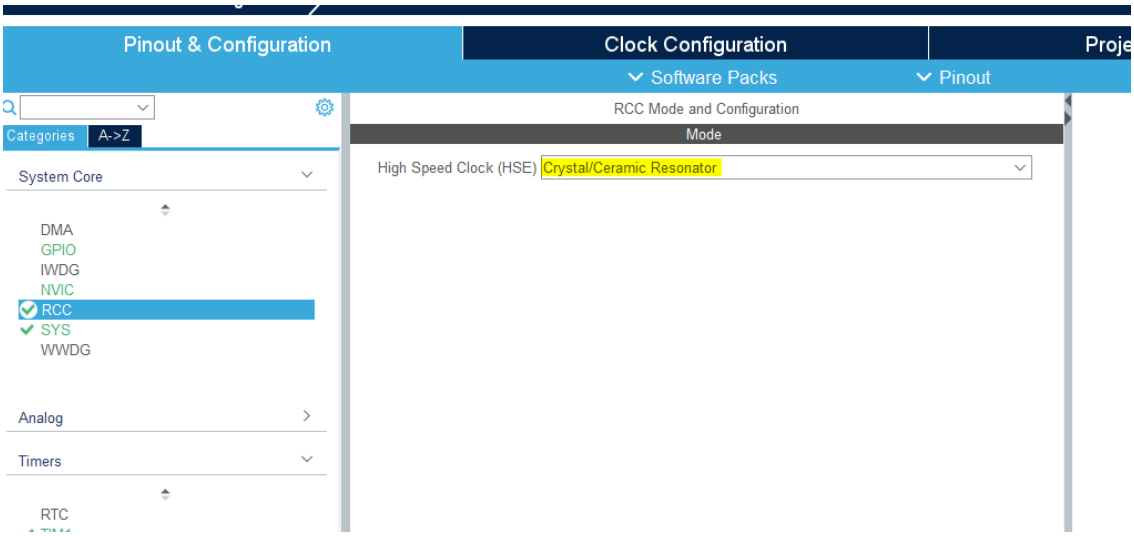
Follow the circuit diagram and hook up the components on the breadboard as shown in the image given below.

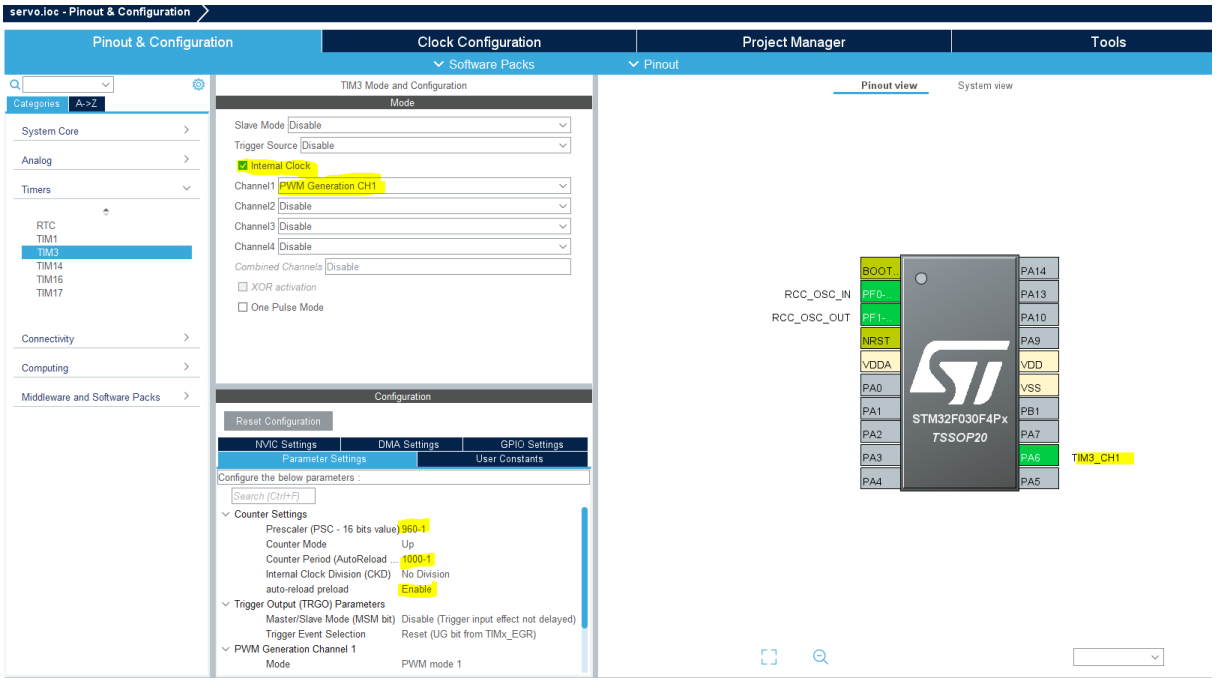


Setup

I am going to use TIM3 of my STM32F0 Dev Board. TIM3 is connected to APB1 bus which gives me timer clock max upto 48 MHz. But the prescalar register is only 16 bit so max value I can write there is 65535. my setup is below:







- As I mentioned above that the frequency must be 50 Hz, it is time to divide the clock using prescalar and the ARR registers. Let's do some calculations than–> (48 MHz/50 Hz) = 960 KHz.
- So now to get a 50 Hz of frequency, I have to divide this 960 KHz between Prescalar and ARR. So I am going to write 960 to prescalar Register and 1000 to ARR.
- The reason behind this is this 1000 will also act as 1000% pulse width and if I want to change the pulse width to any other value, all I have to do is write X% to CCR1 register. This simplifies the things a lot.
- For example, If I want to give a pulse width of 1 ms i.e.$\{(T_c * ARR)/T_{total}\}$ = (1*1000/20) = 50%, I will write 50 instead of X in CCR1 register. For 2 ms, It will be 100%, and for 1.5 ms, It will be 75% and so on.
- After testing it from 1 ms to 2ms, I found that the servo was not rotating the full 180 degrees as it should So I decided to take it a little further and tested it from 0.5 ms to 2.5 ms and It performed well.

Code:

```
#include "main.h"

TIM_HandleTypeDef htim3;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM3_Init(void);

uint16_t i;

int main(void)
{
  HAL_Init();

  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM3_Init();

  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);

  TIM3->CCR1 = 25;  // duty cycle is .5 ms equal to 0 degree

  while (1)
  {

//       TIM3->CCR1 = 25;  // duty cycle is .5 ms equal to 0 degree
//       HAL_Delay(2000);
//       TIM3->CCR1 = 75;  // duty cycle is 1.5 ms equal to 90 degree
//       HAL_Delay(2000);
//       TIM3->CCR1 = 125;  // duty cycle is 2.5 ms  equal to 180 degree
//       HAL_Delay(2000);

        for(i = 25; i<=125; i++){

              TIM3->CCR1 = i;
              HAL_Delay(50);
        }
        for(i = 125; i>= 25;i--){

              TIM3->CCR1 = i;
              HAL_Delay(50);
        }

        HAL_Delay(50);

  }
}
```

- If you remember above, I chose the ARR value of 1000.
- That means the CCR of 25 will be = (CCR/ARR)*100% = (25/1000)*100% = 2.5 % duty cycle. Also remember the cycle time for the pulse is 20ms.
- Now 2.5% of 20ms = (2.5/100)*20 ms = 0.5ms.
- This 0.5ms will correspond to a rotation of 0 degrees.
- Similarly CCR of 75 = 7.5% duty cycle = 7.5% of 20ms = 1.5ms HIGH Pulse. This will correspond to 90 degree rotation
- And at last, a CCR of 125 = 12.5% duty cycle = 12.5% of 20ms = 2.5ms HIGH Pulse. This will correspond to 180 degree rotation